基础类型

- 布尔值
  - 最基本是数据类型就是简单的true/false, 类型定义 `boolean`

```
let isDone: boolean = false
```

- 数字
  - typescript 里的所有数字都市浮点数, 这些浮点数的类型是 `number`
  - 支持十进制和十六进制
  - 支持二进制和八进制

```
let decLiteral: number = 6
let hexLiteral: number = 0xf00d
let binaryLiteral: number = 0b1010
let octalLiteral: number = 0o744
```

- 字符串
  - 我们使用双引号和单引号来表示字符串, 类型定义 `string`
  - 还可以使用反引号来表示字符串

```
let name: string = "blob"
let name: string = `blob`
```

- 数组
  - 在元素后面接上 [] 表示数组
  - 使用数组泛型, Array<数组类型>

```
let list: number[] = [1,2,3]
let list: Array<number> = [1,2,3]
```

- 元组 Tuple
  - 元组类型允许表示一个已知元素数量和类型的数组, 各元素类型不必相同, 比如, 可以定义一对值为 `string` 和 `number` 类型的元组

```
let x: [string, number] = ['hello', 10] // ok
x = [10, 'hello'] // error

x[0].substr(1)
x[1].substr(1) // error, 'number' does not have 'substr'
```

  - 当访问一个越界的元素, 会使用联合类型替代

```
x[3] = 'word' // ok, 字符串可以复制给(string|number)类型
```

- 枚举

- `enum` 类型是对 JavaScript 标准数据类型的一个补充

```
enum Color {Red, Green, Blue}
let c: Color = Color.Green
```

- 默认情况下从 0 开始为元素编号, 你也可以手动的指定成员的数值

```
enum Color {Red=1, Green, Blue}
// or
enum Color {Red=1, Green=2, Blue=4}
```

- 枚举类型提供一个便利是你可以由枚举的值得到他的名字

```
enum Color {Red=1, Green, Blue}
let colorName: string = Color[2]
console.log(colorName) // Green
```

- Any
  - 表示任意值

```
let notSure: any = 4
notSure = 'string'
notSure = false
```

- Void
  - 从某种程度上来说, void 类型像是与 any 类型相反, 他表示没有任何类型, 当一个函数没有返回值时, 你通常会见到其返回值类型是 void

```
function warnUser(): void {
    console.log('this is my warning message')
}

// 声明一个 void 类型的变量只能为他赋予 undefined 和 null
let unuseable: void = undefined
```

- Null 和 Undefined

```
let u: undefined = undefined
let n: null = null
```

- Never
  - never 类型表示的是那些永不存在的值的类型. 如, never 类型是那些总是会抛出异常或根本就不会有返回值的函数表达式或箭头函数表达式返回值类型, 变量也可能是 never 类型, 当他们被永不为真的类型保护所约束时
  - never 类型是任何类型的子类型, 也可以赋值给任何类型, 然而, 没有类型是 never 的子类型或可以赋值给 never 类型, 除了其本身

```
// 返回 never 的函数必须存在无法到达的终点
function error(message: string): never {
    throw new Error(message)
}
```

```
// 推断的返回类型为 never
function fail(){
    return error('something failed')
}

// 返回 never 的函数必须存在无法达到的终点
function infiniteLoop():never {
    while(true) {}
}
```

- Object
  - object 表示非原始类型, 也就是除了 number, string, symbol, null 和 undefined 之外的类型

```
declare function create(o: object | null): void;
create({prop:0})
create(null)
create(42) // error
create('string') // error
```

- 类型断言
  - 使用尖括号

```
let someValue: any = 'this is a string'
let strLength: number = (<string>someValue).length
```

  - 使用 as , 当在 typescript 里使用 JSX 时, 只有 as 语法断言是被允许的

```
let someValue: any = 'this is a string'
let strLength: number = (someValue as string).length
```